# L6 - Objects and Debugging

February 11, 2020

## 1 More Objects and Debugging

### 1.1 Assignment 2

Implement a clock:

- hours
- minutes
- seconds

### 1.2 `this` keyword

`this` is a reference to the current object. `this` is generally used in constructors to disambiguate between class fields and constructor parameters (as an example).

Compare `self` in Python.

[12]:
```
class Cat
{
    // class fields
    private String name;
    private String colour;

    public Cat(String name, String colour)
    {
        this.name = name;
        this.colour = colour;
    }

    public void printInfo()
    {
        System.out.println("LOOK IT'S A KITTY!");
        System.out.println("Its name is " + this.name); // not really necessary␣
 ↪here but why not
        System.out.println("Its colour is " + this.colour);
    }
}
```

[13]:
```
Cat firstCat = new Cat("Socks", "Black and White");
```

[14]:
```
firstCat.printInfo()
```

```
LOOK IT'S A KITTY!
Its name is Socks
Its colour is Black and White
```

### 1.3 Mail System

### 1.4 Mail item class

[5]:
```
/**
 * A class to model a simple mail item. The item has sender and recipient
 * addresses and a message string.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2016.02.29
 */
public class MailItem
{
    // The sender of the item.
    private String from;
    // The intended recipient.
    private String to;
    // The text of the message.
    private String message;

    /**
     * Create a mail item from sender to the given recipient,
     * containing the given message.
     * @param from The sender of this item.
     * @param to The intended recipient of this item.
     * @param message The text of the message to be sent.
     */
    public MailItem(String from, String to, String message)
    {
        this.from = from;
        this.to = to;
        this.message = message;
    }

    /**
     * @return The sender of this message.
     */
    public String getFrom()
    {
        return from;
```

```
    }

    /**
     * @return The intended recipient of this message.
     */
    public String getTo()
    {
        return to;
    }

    /**
     * @return The text of the message.
     */
    public String getMessage()
    {
        return message;
    }

    /**
     * Print this mail message to the text terminal.
     */
    public void print()
    {
        System.out.println("From: " + from);
        System.out.println("To: " + to);
        System.out.println("Message: " + message);
    }
}
```

### 1.5 Mail Server

[6]:
```
import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;

/**
 * A simple model of a mail server. The server is able to receive
 * mail items for storage, and deliver them to clients on demand.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2016.02.29
 */
public class MailServer
{
    // Storage for the arbitrary number of mail items to be stored
    // on the server.
    private List<MailItem> items;
```

```java
/**
 * Construct a mail server.
 */
public MailServer()
{
    items = new ArrayList<>();
}

/**
 * Return how many mail items are waiting for a user.
 * @param who The user to check for.
 * @return How many items are waiting.
 */
public int howManyMailItems(String who)
{
    int count = 0;
    for(MailItem item : items) {
        if(item.getTo().equals(who)) {
            count++;
        }
    }
    return count;
}

/**
 * Return the next mail item for a user or null if there
 * are none.
 * @param who The user requesting their next item.
 * @return The user's next item.
 */
public MailItem getNextMailItem(String who)
{
    Iterator<MailItem> it = items.iterator();
    while(it.hasNext()) {
        MailItem item = it.next();
        if(item.getTo().equals(who)) {
            it.remove();
            return item;
        }
    }
    return null;
}

/**
 * Add the given mail item to the message list.
 * @param item The mail item to be stored on the server.
```

```
     */
    public void post(MailItem item)
    {
        items.add(item);
    }
}
```

### 1.6 Mail Client

[7]:
```
/**
 * A class to model a simple email client. The client is run by a
 * particular user, and sends and retrieves mail via a particular server.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2016.02.29
 */
public class MailClient
{
    // The server used for sending and receiving.
    private MailServer server;
    // The user running this client.
    private String user;

    /**
     * Create a mail client run by user and attached to the given server.
     */
    public MailClient(MailServer server, String user)
    {
        this.server = server;
        this.user = user;
    }

    /**
     * Return the next mail item (if any) for this user.
     */
    public MailItem getNextMailItem()
    {
        return server.getNextMailItem(user);
    }

    /**
     * Print the next mail item (if any) for this user to the text
     * terminal.
     */
    public void printNextMailItem()
    {
        MailItem item = server.getNextMailItem(user);
```

```
        if(item == null) {
            System.out.println("No new mail.");
        }
        else {
            item.print();
        }
    }

    /**
     * Send the given message to the given recipient via
     * the attached mail server.
     * @param to The intended recipient.
     * @param message The text of the message to be sent.
     */
    public void sendMailItem(String to, String message)
    {
        MailItem item = new MailItem(user, to, message);
        server.post(item);
    }
}
```

### 1.7 Debugger

The debugger is a tool to help examine how a program executes. The debugger will help us troubleshoot *logical errors*, not *syntax errors* or other compile errors; the program must actually run in order to be debugged. The debugger will let us read out program variables at various *breakpoints* we can set in the program.

BlueJ's debugger has five panes:

- *instance variables*: the fields of an object
- *static variables*: constants marked with the `static` keyword; probably global
- *local variables*: variables that are part of a method call
- *call sequence*: a *stack* of the methods called. The one on top is the one currently executing.

`printf` debugging isn't gonna work here. Actually debug this thing in BlueJ. Instructions are in the textbook at the end of chapter 3.

[15]:
```
MailServer ms = new MailServer();
MailClient sophie = new MailClient(ms, "sophie");
MailClient juan = new MailClient(ms, "juan");
```

[16]:
```
sophie.sendMailItem("juan", "catch this potato");
```

[17]:
```
juan.printNextMailItem();
```

```
From: sophie
```

```
To: juan
Message: catch this potato
```

[18]:
```
juan.printNextMailItem();
```

```
No new mail.
```

[20]:
```
juan.sendMailItem("sophie", "epstein didn't kill himself");
```

[21]:
```
sophie.printNextMailItem();
```

```
From: juan
To: sophie
Message: epstein didn't kill himself
```

[ ]:
```

```